Cost-Aware Formal Optimization of Finite-Field Arithmetic with `eggstraction`

Matthew P   Praneeth B

CS 292C — Computer-Aided Reasoning for Software

June 5, 2025

GitHub: github.com/realmatthewpeng/eggstraction

Real-World Motivation

- **Cryptographic protocols rely heavily on finite field arithmetic.**
  Pairing-based cryptography, zkSNARKs, and elliptic curve operations all depend on efficient computation in extensions like $\mathbb{F}_{p^{12}}$ and $\mathbb{F}_{p^4}$.[1]
- **Manual optimization is error-prone and ad hoc.**
  Hand-crafted formulas using Montgomery arithmetic, Karatsuba multiplication, and algebraic identities can achieve significant speedups, but are difficult to verify and generalize across different cost models.
- **Optimization depends critically on the target platform.**
  The same finite field operation may favor different implementations depending on whether general multiplication, squaring, or constant multiplication is cheaper—requiring platform-specific manual tuning.
- **Take-away.**
  We need *automated*, *formally verified* optimization that can adapt to different cost models while guaranteeing semantic equivalence.

[1] Beuchat et al. 2010.

Problem Statement & Specification

Given a finite field expression E over $\mathbb{F}_{p^k}$ and cost model C:

Goal

Automatically produce E' such that

$$E' \equiv E \quad \textbf{and} \quad \text{cost}(E'; C) \leq \text{cost}(E; C).$$

Key Challenge

Subexpression sharing matters for cost!

Baseline & Prior Art

Approach	Automated?	Sharing-aware?	Verified?
Hand-optimized formulas (BGMO'10)	$\times$	$\checkmark$	$\times$
Equality saturation (egg)	$\checkmark$	$\times$	$\checkmark$
Computer algebra systems	$\checkmark$	*limited*	$\times$
Our approach	$\checkmark$	$\checkmark$	$\checkmark$

The Gap

Existing equality saturation tools use *greedy tree extraction*, which misses opportunities for subexpression sharing and can lead to unnecessary cost overhead.

Our Contribution

DAG-aware extraction via Integer Linear Programming that optimally accounts for sharing while maintaining formal guarantees.

Key Technical Innovation

Traditional Approach
Equality saturation +
Greedy tree extraction

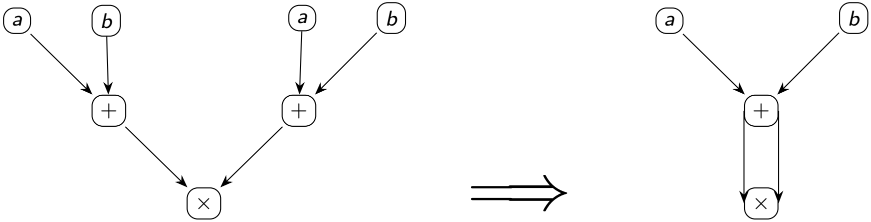
$\Rightarrow$

Our Approach
Equality saturation +
ILP-based DAG extraction

Problem:
Ignores subexpression sharing $\Rightarrow$ Suboptimal costs

Solution:
Optimal extraction with full sharing awareness

Tree vs. DAG Extraction — Why Sharing Matters



TREE extractor
counts $2 \times (+)$

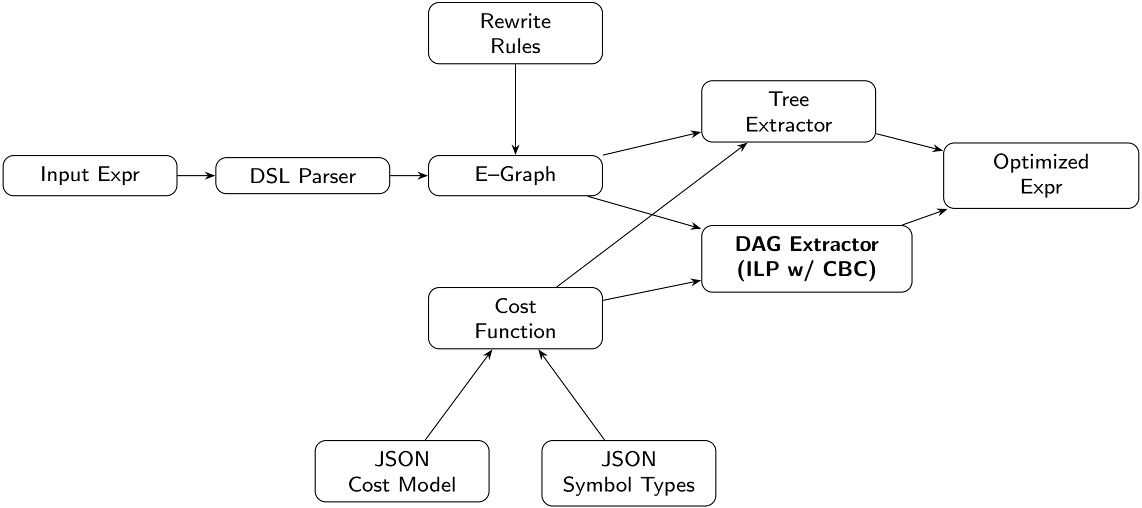
Cost: $2\ (+) + 1\ (\times)$

DAG extractor
shares $1 \times (+)$

Cost: $1\ (+) + 1\ (\times)$

Example expression: `(* (+ a b) (+ a b))`.

System Architecture



Walk-Through on Motivating Example

Input Expression

```
(pair (+ (sq a0) (* (sq a1) xi)) (* 2 (* a0 a1))
```

1. **Input** $(a0 + a1V)^2$ where $a0, a1 \in Fp_2$: naive cost $= 31$.
2. Apply binomial rewrite rule in *egg* egraph.
3. Perform DAG extraction: identify shared $\text{sq}(a0)$, $\text{sq}(a1)$.
4. **Output** cost $= 26$ $(-16\,\%)$.

Optimized Expression

```
(pair (+ (sq a0) (* (sq a1) xi)) (- (- (sq (+ a0 a1)) (sq a1)) (sq a0)))
```

Implementation Highlights

- E-Graph framework: *egg*.
  - Configurable cost function (read from JSON).
  - DSL: basic arithmetic, pair syntax to model tower fields.
  - **TypeAnalysis**: tracks field extension degree via LCM.
  - Basic Tree extractor.
- DAG extractor: https://github.com/egraphs-good/extraction-gym.
  - ILP approach using CBC solver.
  - Preprocesses egraph for faster solving.
- Python wrapper script enabling finite field towering construction.

Evaluation Setup

Benchmarks BGMO Algorithms 5–31.

Cost Models Default (Appendix II).

Machine 2019 Intel MacBook Pro.

Results

10 benchmarks replicated

Benchmark	Description	Naive Cost	**Our Cost**
5	Fp_2 Addition	1	0.2*
6	Fp_2 Subtraction	1	0.2*
7	Fp_2 Multiplication	4	3*
9	Fp_4 Squaring	27	26*
10	Fp_6 Addition	3	3
11	Fp_6 Subtraction	3	3
14	Fp_6 Multiplication	30	30
18	Fp_{12} Addition	12	12*
19	Fp_{12} Subtraction	12	12*
20	Fp_{12} Multiplication	310	470*

* Discovered with automated towering construction

Benchmark 20 Case Study

Input: $A = a_0 + a_1 w,$ and $B = b_0 + b_1 w \in \mathbb{F}_{p^{12}}$

Output: $C = c_0 + c_1 w = A \cdot B \in \mathbb{F}_{p^{12}}$

1. $t_0 \leftarrow a_0 \cdot b_0$ Cost: 130
2. $t_1 \leftarrow a_1 \cdot b_1$ Cost: 130
3. $c_0 \leftarrow t_0 + t_1 \cdot \gamma$ Cost: 56
4. $c_1 \leftarrow (a_0 + a_1) \cdot (b_0 + b_1) - t_0 - t_1$ Cost: 154
5. **return** $C = c_0 + c_1 w$

Total Cost: 470

Cost of Multiplication in Fp_{12}: 310

Limitations & Future Work

- Optimization requires non-trivial rewrite rules.
  - Add support for more operations: inverse, exponentiation, etc.
- Extraction is NP-Complete: solver is slow, especially with egraph blowup.
  - Use LLMs to choose minimal set of necessary rewrite rules.
- Better support for towering finite fields.
  - Currently, can only synthesize quadratic extensions.
- Cannot handle programs with loops.

Take-Away

Eggstraction can automatically and correctly optimize simple finite-field computations with respect to a given cost model.

Live Demo

`cargo run`